DEBUG (DEBUG/DEB32)

MANUAL HISTORY

Manual Order Number: 211-804013-E02

Title: MAX IV AND MAX 32 PROGRAMMER'S REFERENCE MANUAL, DEBUG (DEBUG/DEB32)

Software Part Numbers: 610551-000E.1 and 610XXX-000A.0

Revision Level	Date Issued	Description
---	08/76	Initial Issue (A.0).
---	06/78	Reissue.
D00	07/79	Reissue (D.0).
E00	12/83	Revision (E.0). Support documentation updates for revision E.0.
E01	08/84	Reissue (E.0). Reformatted manual to publication standards.
E02	07/85	Reissue (E.1 and A.0). MAX 32 Debug information was included.

Contents subject to change without notice.

Copyright ©1976, by Modular Computer Systems, Inc.
All Rights Reserved.
Printed in the United States of America.

PROPRIETARY INFORMATION

THIS MANUAL CONTAINS CONFIDENTIAL PROPRIETARY INFORMATION PROTECTED BY SOFTWARE PROPERTY RIGHTS AND IS MADE AVAILABLE UPON THE CONDITION THAT THE SOFTWARE CONTAINED HEREIN WILL BE HELD IN ABSOLUTE CONFIDENCE AND MAY NOT BE DISCLOSED IN WHOLE OR IN PART TO OTHERS WITHOUT THE PRIOR WRITTEN PERMISSION OF MODULAR COMPUTER SYSTEMS, INC.

FOR GOVERNMENT USE THE FOLLOWING SHALL APPLY

RESTRICTED RIGHTS LEGEND

USE, DUPLICATION OR DISCLOSURE BY THE GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN PARAGRAPH (b) (3) (B) OF THE RIGHTS IN TECHNICAL DATA AND COMPUTER SOFTWARE CLAUSE IN DAR7-104.9(a) (OR SUCH EQUIVALENT FPR, NASPR, AND/OR DOEPR CLAUSE AS MAY BE APPLICABLE)

MODULAR COMPUTER SYSTEMS, INC.
1650 WEST McNAB ROAD
FORT LAUDERDALE, FL 33309

PREFACE

Audience This manual is directed to programmers who create or maintain source programs.

Subject This manual describes the functional operation and use of MAX IV and MAX 32 Debug.

Product MAX IV and MAX 32 Debug (referred to as DEBUG/DEB32) are MAX IV and MAX 32 system processors that assist in the debugging of machine language programs on MODCOMP CLASSIC compatible computers. The use of the term **machine language** is not meant to imply that DEBUG/DEB32 can only be used on an Assembly Language program (although that is their major function). DEBUG/DEB32 provide debugging facilities in terms of machine instructions and memory addresses rather than in terms of symbolic labels and source line numbers. The DEBUG/DEB32 processors are Job Control overlays.

How to Use This Manual

Chapter 1 contains summary information describing DEBUG/DEB32 and their usage.

Chapter 2 describes the commands in a reference format.

Chapter 3 contains a detailed description of all commands in alphabetical order.

Chapter 4 lists messages generated for errors and expected conditions.

Chapter 5 summarizes the output of the various DEBUG/DEB32 functions.

Related Publications Refer to the following manuals for additional information. When ordering manuals, use the manual order number listed below. The most current revision level (REV) will be shipped.

Manual Order Number	Manual Title
211-804005-REV	MAX IV SOURCE EDITOR Programmer's Reference Manual (PRM)
213-804002-REV	MAX IV SYSGEN System Guide Manual (SGM)
213-838002-REV	MAX 32 SYSGEN System Guide Manual (SGM)
213-804004-REV	MAX IV OPERATOR COMMUNICATIONS System Guide Manual (SGM)
213-838004-REV	MAX 32 OPERATOR COMMUNICATIONS System Guide Manual (SGM)
211-804002-REV	MAX IV/MAX 32 NONRESIDENT JOB CONTROL AND BATCH FACILITIES Programmer's Reference Manual (PRM)

226-723001-REV	CLASSIC II Series Central Processor Single Board, Computer Reference Manual (XRM)
226-720001-REV	CLASSIC Central Processor CLII/25 Computer Reference Manual (XRM)
226-721001-REV	CLASSIC II Central Processor CLII45/55 Computer Reference Manual (XRM)
226-722001-REV	CLASSIC Central Processor Model CLII/75 Computer Reference Manual (XRM)
226-724001-REV	CLASSIC 32/85 32-Bit Machine Mode Computer Reference Manual (XRM)
226-724002-REV	CLASSIC 32/85 16-Bit Machine Mode Computer Reference Manual (XRM)
228-724001-REV	CLASSIC 32/85 General Theory Manual (GTM)

MODCOMP Product Training

MODCOMP's Education Services provides training courses on many hardware and software products at our Training Center in Ft. Lauderdale, Florida. On-site courses at customer sites can also be arranged. For more information about the courses offered by Education Services, contact the MODCOMP Training Registrar.

MAX IV/MAX 32 SUMMARY

This revision of the manual includes information for both MAX IV DEBUG and MAX 32 DEB32. The user interface for DEBUG and DEB32 is the same except for output format (shown in Chapter 5). Changes include the following:

- References to extended memory are no longer supported under DEB32. Extended memory applies only to:

  - CLASSIC II/75
  - CLASSIC 32/85 16-bit machine mode
  - CLASSIC 32/85 32-bit machine mode, 16-bit user mode

- The MODIFY Directive includes additional information that explains how to modify using either absolute values or addresses for 32-bit values.

TABLE OF CONTENTS

Page

CHAPTER 1 DEBUG/DEB32 OVERVIEW 1-1

1.1 GENERAL INFORMATION 1-1
1.2 DEBUG/DEB32 PROGRAM STATES 1-2
1.3 COMMAND FORMATS AND PARAMETERS 1-3
1.4 ADDRESSING 1-3
1.5 DISPLAY OF ADDRESSES 1-4
1.6 LOADING AND BIASES 1-4
1.7 LOADING AN ABSOLUTE MODULE 1-4
1.8 DISPLAY COMMANDS 1-5
1.9 MODIFY COMMAND 1-5
1.10 SET COMMAND 1-6
1.11 EXECUTION CONTROL 1-6
1.12 BREAKPOINTING 1-7
1.13 TRAPS 1-7
1.14 SNAPSHOTTING 1-8
1.15 TRACING 1-8
1.16 STEPPING 1-10
1.17 LIMITATIONS 1-10

CHAPTER 2 REFERENCE INFORMATION 2-1

2.1 FILES 2-1
2.2 PARAMETERS 2-1
2.3 DEBUG/DEB32 COMMAND ADDRESSES 2-2

CHAPTER 3 COMMANDS 3-1

ACTIVATE - Mark Trace As Active 3-2
BIAS - Set (Or Display) Address Relocation Bias 3-3
BTRACE - Set Branch Trace 3-4
CLEAR - Remove Entries From DEBUG/DEB32 Table 3-5
DEFINE - Equate Value To A Symbol 3-6
DISPLAY - Exhibit Memory Or Register Content 3-7
EXIT - Leave DEBUG/DEB32 Processor 3-8
GOTO - Start (Or Restart) Program Execution 3-9
HELP - Display DEBUG/DEB32 Usage Information 3-10
INPUT - Set Input Mode 3-11
LOAD - Get Program To Be Debugged 3-12
LOCATE - Set (Or Display) Load Bias 3-13
MODIFY - Modify Contents Of Register(s) Or Memory 3-14
NOP - No Operation 3-16
OPTION - Set DEBUG/DEB32 Options 3-17
OUTPUT - Set Output Modes 3-18
PAUSE - Set Breakpoint In User Program 3-19
REFERENCE - Set Reference Trace 3-20
RESET - Deactivate All Active Traces 3-21
SET - Set Memory Array To Constant 3-22
SHOW - Display DEBUG/DEB32 Table Contents 3-23
SNAP - Set Snapshot 3-24

STEP - Step Through Program 3-25
TRACE - Set A Trace 3-26
TYPE - Display Values On CO File 3-27

CHAPTER 4 MESSAGES 4-1

CHAPTER 5 OUTPUT FORMATS 5-1

INDEX I-1

CHAPTER 1
DEBUG/DEB32 OVERVIEW

A large part of the software development effort is consumed in dealing with errors. There are three basic types of software development errors:

- DESIGN ERRORS - Program design does not fulfill the concept of the solution to the problem.
- IMPLEMENTATION ERRORS - Actual program does not correctly implement the design.
- CONCEPT ERRORS - Specified requirements do not solve the problem or the concept of the solution does not meet the requirements.

Design errors and implementation errors are commonly referred to as **bugs** and the process of finding and eliminating them is called **debugging.** The state of the art has not yet provided significant tools to assist in dealing with concept errors, however, a number of tools have been developed to assist in the debugging process.

MAX IV Debug (DEBUG) and MAX 32 Debug (DEB32) are referred to as DEBUG/DEB32 throughout this manual. DEBUG and DEB32 are system processors that assist in the debugging of machine language programs on MODCOMP CLASSIC compatible computers. The use of the term **machine language** is not meant to imply that DEBUG/DEB32 can only be used on an Assembly language program (although that is their major function). DEBUG/DEB32 provide debugging facilities in terms of machine instructions and memory addresses rather than in terms of symbolic labels and source line numbers.

1.1 GENERAL INFORMATION

The DEBUG/DEB32 processors are Job Control overlays. They are cataloged under the names DEBUG (for MAX IV on BM) and DEB32 (for MAX 32 on VBM). DEB32 does not have extended memory. Extended memory applies only to:

- CLASSIC II/75
- CLASSIC 32/85 16-bit machine mode
- CLASSIC 32/85 32-bit machine mode, 16-bit user mode

Output format for DEBUG and DEB32 varies slightly. Addresses are shown in a 32-bit quantity when applicable (Example: 32-bit display registers and trace addresses). Due to this, the trace line for DEB32 is output in two or three lines instead of one or two.

The DEBUG/DEB32 processors are designed to be used interactively. The programmer enters commands through a terminal and observes the results in a random sequence. The DEBUG/DEB32 processors can be used in a batch environment using a fixed job stream, but the debugging effectiveness of this method is low.

1.2 DEBUG/DEB32 PROGRAM STATES

DEBUG/DEB32 are always in one of two states:

- Command state
- Execution state

In command state, DEBUG/DEB32 will accept and process user commands. In execution state, the user program is executing, whether under DEBUG/DEB32's direct control or not. The transition from command state to execution state is accomplished by using the GOTO Command (GOT or G). Transition from execution state to command state occurs when a breakpoint is encountered or when a violation or error occurs.

Program registers and memory are displayed or modified in the command state. The display can be on the user's terminal or on the DO file (normally a high speed printer). The values are displayed in hexadecimal, decimal, ASCII, or CAN formats. Registers, memory, and extended memory (see note below) can be modified with values that are in hexadecimal, decimal, or CAN only.

NOTE: Extended memory applies only to: CLASSIC II/75, CLASSIC 32/85 16-bit machine mode, and CLASSIC 32/85 32-bit machine mode, 16-bit user mode.

Memory can also be modified by a patch assembler facility. Arrays of memory and extended memory (see note above) can be set to a single value.

A series of commands is available allowing the user to exercise control over the user program while it is in the execution state.

A set of commands allowing the user to control DEBUG/DEB32 includes:

- Commands that set various options
- Commands that set relocation biases
- Commands that display the current state of various DEBUG/DEB32 parameters

Commands are available to:

- Provide file control
- Prepare user programs
- Start user program execution
- Leave DEBUG/DEB32

1.3 COMMAND FORMATS AND PARAMETERS

DEBUG/DEB32 accept commands as three character keywords. Any characters beyond the first three in the keyword are ignored. Several commands are acceptable in a single character form. Two character commands are <u>not</u> acceptable. All characters in commands and parameters can be upper or lower case.

Several formats of input and output are provided. The INPUT (INP) and OUTPUT (OUT) Commands set the modes to be used.

1.4 ADDRESSING

Many commands require one or more address parameters. These addresses can be interpreted in either of two ways:

- ABSOLUTE - The address value is interpreted as an absolute address (relative to the start of the address space).
- RELATIVE - The address value is interpreted as being relative to the current relocation bias that defaults to the BIAS Command.

Absolute addresses are needed when displaying or modifying linked lists and other data structures that contain address pointers.

Relative addresses are used for manipulating instructions in the user program, for setting execution control, and for manipulating normal data items. A relative address is specified by preceding the address with a dollar sign ($).

EXAMPLE

$123 or $#2A7

NOTE: Since relative addresses are the most commonly used, an option of the INP(UT) Command causes all input addresses to be interpreted as being relative, without the need to enter the $. This option is the default at startup. In any event, if a range of addresses is specified, and the first address is relative, the second is automatically relative.

Numbers can be input in either decimal or hexadecimal form. The default at startup is decimal. Hexadecimal numbers are indicated with the hexadecimal flag (#).

EXAMPLES (Assume startup default)

123 Represents decimal
#123 Represents hexadecimal
12A Represents an error

NOTE: Since hexadecimal values are commonly used for addresses, it is possible to set hexadecimal as the default input mode using the INP(UT) Command. In this case, a decimal number can be specified by preceding it with the decimal flag (.).

EXAMPLES (Assume INP HEX Command sets default input mode to hexadecimal)

123 Represents hexadecimal
#123 Represents hexadecimal
A27 Represents hexadecimal
.123 Represents decimal
.1A7 Represents an error

1.5 DISPLAY OF ADDRESSES

As mentioned, input addresses can be absolute or relative to the current value of the relocation bias. Similarly, the form used for displaying addresses depends on the current setting of the ABSOLUTE or RELATIVE parameter of the OUT(PUT) Command. If the command OUT(PUT) ABS(OLUTE) is given, regardless of context, all addresses displayed are displayed without modification.

If the command OUT(PUT) REL(ATIVE) is given (the default at startup), all addresses (that can definitely be determined to be addresses) are displayed as values relative to the current relocation bias.

1.6 LOADING AND BIASES

To prepare a program for debugging, the LOAD Command (LOA) is used. This command loads the user program into memory, and allows the user to specify options in the same manner as the Job Control $EXECUTE Command.

DEBUG/DEB32 assume that all programs that are worked on are cataloged by the MAX IV or MAX 32 Task/Overlay Cataloger (TOC/TOC32) as Peculiar Relocatable (PEC REL). Normally, if the loaded program is not relocatable an error message is displayed and the module is not usable.

1.7 LOADING AN ABSOLUTE MODULE

An absolute module can be debugged if it was biased at link edit time. The LOC(ATE) Command gives the current load bias if you forget what the startup message said. Absolute modules must be loaded above DEBUG/DEB32's memory space.

CAUTION

The load bias value must be set to zero for an absolute module. DEBUG/DEB32 must not check the safety of the LOCATE Command's parameter. DEBUG/DEB32 are destroyed if a low value (or zero) is specified and the actual module loaded is not biased past DEBUG/DEB32. DEBUG/DEB32 must **not** check whether the module loaded is relocatable. The option NOC (no check) accomplishes this. The necessary sequence is:

```
OPT NOC
LOC 0
LOA the module
LOC X (where X is the actual bias)
OPT CHE (to restore the normal state)
```

A second bias value can be set using the BIAS Command (BIA or B). This value specifies the relocation value used for relative addresses and has no other purpose. It is independent of the load bias, except that a LOCATE Command sets the relocation bias to the new load bias. A BIAS Command with no operand resets the relocation bias to the current load bias.

1.8 DISPLAY COMMANDS

TYPE and DISPLAY Commands are a pair of commands used to display the contents of memory, extended memory (CLASSIC II/75 only), and registers.

The TYPE Command (TYP or T) displays information on the CO file (normally the programmer's terminal). The DISPLAY Command (DIS or D) displays this information on the DO file (normally assigned to a high speed printing device).

Other than directing their output to different files, the syntax and actions of these two commands are the same.

Each of the commands above can take an optional format specifier. Those provided are:

H - hexadecimal
D - decimal
A - ASCII
C - CAN (actually, unCAN)

EXAMPLE

T C $#123 10 - Displays from #123 for ten words, in CAN format. If no format specifier is included, the current default (set by the OUTPUT Command) is used.

1.9 MODIFY COMMAND

The MODIFY Command (MOD or M) allows memory, extended memory and registers to be modified. More than a single register or memory location can be modified by a single command. A patch assembler facility is available for modifying memory only.

To modify memory, the start address is specified followed by a series of values that are placed in that start address memory location and succeeding memory locations.

The patch assembler allows the user to modify memory using assembler mnemonics instead of doing a translation. The patch assembler requires specification of all instruction subfields. Those subfields that are augments, (not directly specified by the user but are inherent in the instruction) are shown in the appropriate CPU reference manual. These subfields should be specified as null. A null field is one that is omitted and can be indicated by either a pair of consecutive commas (,,) or by completely omitting a trailing item. Refer to the MODIFY Command in Chapter 3 of this manual for more information.

EXAMPLES

M $#20 LDM*,4,0,$#1022
- Places into memory at relative #20 and #21 an LDM instruction with the indirect flag set, with no indexing, and with an address of relative #1022.

M $#37 BLHI,,4,$#123
- Places the BLHI instruction with a link register of 4 into location #37, and the branch address of #123 into location #38.

Extended memory modification is analogous to normal memory modification without the assembler facility.

EXAMPLE (DEBUG only)

M !#123 #1234,@ABC,787
- Places #1234, @ABC, and 787 (decimal) into extended memory locations #123, #124, and #125, respectively.

Register modification is specified by using the R(EG) option with a pair of values. The first value of the pair is the register to be modified, the second value is the value the specified register is to receive. The assembler facility is not available.

EXAMPLE

M R 3 @DEF 14 #123 9 7
- Stores @DEF into register 3, #123 into register 14, and 7 into register 9.

1.10 SET COMMAND

The SET Command (SET) sets arrays of memory or extended memory to a single value. An address range and a decimal, hexadecimal, or CAN code value must be provided. Extended sets are analogous, with the extended flag specified, and 21-bit addresses are permitted.

1.11 EXECUTION CONTROL

While the user program is in the execution state, DEBUG/DEB32 allow the user to exert control over it in several ways. These controls include setting breakpoints, taking snapshots, and tracing.

Each of the execution control techniques has an associated command:

PAUSE - sets breakpoints
SNAP - sets snapshots

TRACE \
BTRACE } - establish traces
REFERENCE /

These commands are described in detail in the following sections under BREAKPOINTING, SNAPSHOTTING, and TRACING.

Each execution control command can be thought of as a request for later action. The command specifies one or two addresses; when these locations are encountered during execution, some action takes place.

Each execution control command places an entry into tables (called the DEBUG or DEB32 Tables) that provide instruction for DEBUG/DEB32 when addresses are encountered. A single address can appear only once in the table. No more than one trace, breakpoint or snapshot can specify a given address at any given time. The DEBUG or DEB32 table has room for 20 entries at a time. The current entries in the DEBUG or DEB32 table can be displayed using the SHO(W) Command.

Entries can be removed using the C(LE(AR)) Command. A LOAD Command also clears the DEBUG or DEB32 table.

The user program enters the execution state with the GOTO Command after execution control commands are entered. GOTO takes an optional parameter that defines the address location where execution is to begin. The absence of a parameter causes execution to continue from the current set location to the load time entry point.

1.12 BREAKPOINTING

Breakpointing allows the user to specify addresses where DEBUG or DEB32 is to regain control and enter the command state. Breakpoints are set using the P(AU(SE)) Command.

The PAUSE Command is entered before executing the program. When the instruction at location #234 is encountered, the following message is displayed on the terminal:

MESSAGE

PAUSE AT #234

NOTE: The address displayed is dependent on OUT mode whether it is ABS or REL. DEBUG or DEB32 re-enters the command state after the message is displayed.

Breakpoints can be made conditional. Two separate types of conditions can be tested for.

- Counted Pause
- Equal Pause

A count can be specified in the PAUSE Command, causing the breakpoint to occur only every **count-th** time that the address is encountered.

An equal pause is the second form of conditional breakpoint. Two parameters are specified and tested for when the indicated location is encountered. If the parameters are equal, the breakpoint is taken. Each parameter can be a memory location, an extended memory location (CLASSIC II/75 only), a register, a constant, or a bit mask.

The bit mask form does a bit-wise logical AND between the mask and the other parameter before the comparison. Refer to the following example.

EXAMPLE

P $#234 B #FF00 R 6 - Causes a breakpoint at #234 if the value in register 6 ANDed with #FF00 is equal to #FF00. If the value in register 6 was a #FF72, a breakpoint is taken; if it was #F772, a breakpoint is not taken.

1.13 TRAPS

As stated earlier, DEBUG/DEB32 will go from execution state to command state if a breakpoint is taken or if a trap occurs.

A trap is a signal from the operating system that an exceptional condition has occurred. These conditions are:

- an attempt to execute an unimplemented instruction;
- an attempt to execute an unimplemented REX;
- an attempt to execute any REX through the EXR instruction;
- a protect violation of any type (that produces a PRO Abort outside DEBUG/DEB32);
- an attempt to execute a REX EXIT;
- an attempt to execute a REX USERTRAP;
- a floating point exception (overflow or underflow).

In each of these cases a message is displayed indicating the nature of the exceptional condition and the address where the trap occurred.

NOTE: In the case of a floating point exception that is in direct execution, the address displayed is the instruction address following the one causing the exception. Use of stepping or tracing causes the correct address to be displayed.

The floating point exception trap can be enabled or disabled through the use of the OPTION Command. If the floating point exception trap is active, the trap occurs. If not active, the operating system's default action (that sets the result to the closest available hardware representable value) is used.

1.14 SNAPSHOTTING

The user can sample the state of the program at specific points during execution using DEBUG/DEB32. This technique is known as snapshotting, and it is invoked by the SNAP Command (SNA or S).

A snapshot includes a display of all registers. Optionally, memory or extended memory can also be displayed. All SNAP output is directed to the DO file.

The SNAP Command accepts an optional format like the TYPE and DISPLAY Commands. If no explicit format is provided, the default at the time the SNAP Command is given (rather than the default at the time the snapshot takes place).

1.15 TRACING

Tracing is an execution state process. Tracing allows the user to observe the program during execution with a high degree of resolution. While operating in the trace mode, DEBUG/DEB32 execute the instructions of the user program one at a time. After each instruction is executed, the complete state of the machine is available and can be displayed. This state information includes the contents of the registers, the condition codes, and the contents of the effective address, if any exist. Three forms of trace are available.

- TRACE
- BTRACE
- REFERENCE TRACE

All perform the actual trace in the same fashion, but they differ as to when the state information is displayed.

The TRA(CE) Command sets a trace displaying the trace information for each executed instruction. The BTR(ACE) Command (BTR represents **B**ranch **TR**ace) sets a trace displaying the trace information after instructions that cause a branch. The REF(ERENCE) TRACE Command sets a trace displaying the trace information after executing instructions that reference an address that the command specified.

The entire program rarely requires to be traced. Each TRACE Command specifies a range of addresses over which the trace is active. The trace is activated when the first address is encountered and deactivated when the last address is encountered.

NOTE: Only these two addressses are significant.

If the user's program skips over the first instruction, the trace does not become active, even if the program is executing instructions within the trace range. If the trace is active, and the user's program skips over the end address, the trace remains active, even though the program is no longer executing instructions within the trace range.

Traces can be arbitrarily nested or overlapped. Tracing continues as long as any traces are active, however, the output is governed by all active traces. Only one trace line can be displayed although several traces can call for output.

Trace output is directed to the DO file. A complete trace output line is roughly 130 characters, and not all devices can accept a line of this length. Three options (specified by the OUTPUT Command) are available for formatting trace output. These are:

- ONE - Generates a single (long) output line.
- TWO - Splits the line into two shorter pieces.
- INS - Generates a single line, but does not display the registers.

Examples of each option are given in Chapter 5 "OUTPUT FORMATS" of this manual.

Reference traces are set with the REF Command.

It is sometimes desirable to manually turn traces on or off. After reaching a breakpoint in the middle of a trace, it can be determined that it is no longer necessary to trace that part of the program. Providing these capabilities, the A(CT(IVATE)) and R(ES(ET)) Commands are used.

The ACTIVATE Command takes as a parameter an address that must be the start address of a trace. Its action is to make that trace active, regardless of whether the start address has been reached or not. The RESET Command takes no parameters and deactivates all currently active traces. It does not remove active traces from the DEBUG/DEB32 table. These traces can be reactivated either normally or with an ACTIVATE Command. The CLEAR Command removes traces from the DEBUG/DEB32 table. An active trace that is cleared is deactivated first.

1.16 STEPPING

Another very useful debugging technique is single stepping. Instructions are executed one by one, and the machine state is displayed after each instruction. Stepping occurs in command state and is not tied to any fixed set of addresses. Stepping can start at any location.

It is possible to step one or more instructions in a single command. When more than one instruction is to be stepped, a new command cannot be accepted until the specified number of instructions have been executed. For each instruction stepped, information is displayed on the CO file. The display is in a trace output line in the OUTPUT INS format. A multi-instruction step interrupted by a breakpoint or a trap cannot resume, even if a GOTO Command is given.

The STEP Command format begins with a number that specifies the amount of instructions to step, followed by an address that specifies the location where stepping begins. Both parameters are optional.

NOTE: If the address is present, both parameters must be entered. If the address is missing, stepping begins from the current location (the value of the program register). If both parameters are missing (a null command line using a carriage return), a single step is taken from the current location.

1.17 LIMITATIONS

DEBUG/DEB32 are used in debugging single-mapped non-privileged overlays and tasks loaded as overlays. Privileged instructions mnemonics cannot be used in the MODIFY Command, and privileged instructions cannot be traced, paused or snapped on. If DEBUG or DEB32 is cataloged as non-privileged (the normal case), privileged instructions cannot be used.

DEBUG/DEB32's protection against destruction is good, but the use of certain REX services can compromise it. There is no protection against any of the memory map modifying REXes (ALLOCATE, DEALLOCATE, LOVER, MARS, INSGLO, and so on). These REXes affect DEBUG/DEB32's pages rather than those of the user program. The use of absolute page numbers can lead to problems.

Programs using MAX IV File Manager REX services when Error Processing Addresses (EPAs) are used can be debugged. Trace mode does not successfully **follow** the EPA branch. It is recommended that a PAUSE be placed at the EPA address if EPAs are used.

Certain actions that are normally terminal (any violation, REX EXITs, and so forth) are trapped by DEBUG/DEB32 and do not cause the processor to be exited. An explicit REX ABORT in the user program, or any implicit aborts caused by REXes in the user program (I/O or File Manager REXes) causes an abort and the exit of the DEBUG or DEB32 processor.

NOTE: A PAUSE, SNAP, or TRACE start cannot be set on the instruction immediately following an EXMA instruction.

An address can be an effective address for only one DEBUG/DEB32 command at a time.

The following sequence is illegal:

```
P      $4
TRA    $4-17
```

The following sequence is valid:

```
P      $4
  .
  .
  .
TRA    $4-17
```

CHAPTER 2
REFERENCE INFORMATION

This chapter describes each of the available files, file manipulation commands, delimiters, and command addresses.

2.1 FILES

DEBUG and DEB32 use the following files:

- CI - Commands are accepted from this file.
- AI - If the AO option is set when DEBUG or DEB32 is executed, this file is used instead of CI.
- CO - All messages, TYPE and STEP output, and inquiry responses are written to this file. This file is assumed to be assigned to a terminal device.
- DO - DISPLAY, SNAP and TRACE output, and any echoed commands and messages are written to this file.
- Quick load module files - Any quick load module file name can be referenced on a LOAD Command.
- Any file -Any file can be referenced in an ASSIGN Command, or any of the file manipulation commands.

The use of the file manipulation commands as well as their command syntax is the same as in Job Control (except for the leading $). The file manipulation commands are:

- ASSIGN
- AVFILE
- BKFILE
- AVRECORD
- BKRECORD
- HOME
- REWIND, and
- WEOF

Refer to the MAX IV/MAX 32 NONRESIDENT JOB CONTROL AND BATCH FACILITIES, Programmer's Reference Manual (listed in the Preface) for information on the use of these commands.

2.2 PARAMETERS

All command parameters can be separated by blanks or commas unless otherwise stated.

2.3 DEBUG/DEB32 COMMAND ADDRESSES

The address of a PAUSE or SNAP and the start and end addresses of any trace must be the same as that of an instruction in the user program. In the case of multi-word instructions, it must be the address of the first word of the instruction. Since DEBUG/DEB32 are unable to determine if these rules are followed, violations do not produce error messages but produce unexpected results or errors in execution state.

CHAPTER 3
COMMANDS

This chapter presents detailed discussions of each DEBUG/DEB32 command. The discussions are presented in alphabetical order based on the command name. Each discussion includes a functional description of the command, the syntax of the command, and examples where syntax contains more than the command. Keywords at the beginning of each command aid in location.

The format used in the presentation of this information is as follows:

KEYWORD

Brief Description

Description in paragraph form.

SYNTAX

KEYWORD parameter1 [parameter2]

parameter1 (REQUIRED) (Parameter 1 description in paragraph form)

[parameter2] (OPTIONAL) (Parameter 2 description in paragraph form)

Symbols used:

xxx	Lower-case characters indicate that the value of the parameter must be supplied by the user.
XXX	Upper-case keywords are to be entered as shown.
[ ]	Brackets indicate either insignificant characters or an optional parameter.
{ }	Braces indicate a set of options from which the user is required to choose at least one.

ACTIVATE

Mark A Trace As Active

The ACTIVATE Command activates the trace starting at **Addr.** The effect is the same as if program execution had passed through **Addr,** starting the trace normally. The current location is not altered.

SYNTAX

A[CT] addr

addr (REQUIRED) Parameter 1 is the valid location to start a trace.

Set (Or Display) The Address Relocation Bias

The BIAS Command specifies the setting for the relocation bias or displays the current relocation bias.

SYNTAX

B[IA] [addr]
[?]

[addr]
[?]

(OPTIONAL) Parameter 1 specifies one of the following:

addr specifies the setting for the relocation bias.

? request the display of the current relocation bias. This value is relative to the current load bias.

If neither parameter is present, the relocation bias is reset to the current load bias.

Set A Branch Trace

The BTRACE Command sets a branch trace. A branch trace is the same as a normal trace, except that a trace output line is produced only for those instructions that cause a hop or branch.

SYNTAX

BTR start-addr {-end-addr} {count}

NOTE: When using instructions with conditional branches, the trace line is output only if the branch is actually taken.

start-addr (REQUIRED) Parameter 1 specifies the starting address for the branch trace.

{-end-addr} {count} (AT LEAST ONE IS REQUIRED) Parameter 2 specifies how far the trace goes.

-end-addr signals the end of the run.

count specifies the number of words the trace contains.

Remove Entries From The DEBUG/DEB32 Table

The CLEAR Command removes specified DEBUG/DEB32 Table entries.

SYNTAX

C[LE] {ALL}
{addr...}

{ALL}
{addr} (AT LEAST ONE IS REQUIRED) Parameter 1 specifies one of the following:

ALL removes all entries currently in the DEBUG or DEB32 table.

addr list removes the entries for each **addr** location (in trace-type entries this is the start location). If an **addr** doesn't match an entry in the DEBUG/DEB32 table, a message is displayed and processing of the command is stopped. All addresses to the left of the error arrow are processed.

DEFINE

Equate A Value To A Symbol

The DEFINE Directive equates a value to a symbol.

SYNTAX

DEF symbol addr

symbol — (REQUIRED) Parameter 1 associates an address with a symbol. After a symbol is defined it can subsequently be used in any directive instead of a numeric value.

addr — (REQUIRED) Parameter 2 represents the symbol value. The value cannot be in extended memory.

Display On DO The Contents Of Memory, Extended Memory, Or Registers

The DISPLAY Command displays memory or register contents on the DO file.

SYNTAX

```
D[IS]  [format]   {r}        [reg...]
                  {addr}     [-addr
                             [count]
                  {!xaddr}   [-xaddr]
                             [count]
                  {!?}
```

[format] (OPTIONAL) Parameter 1 specifies one of the following formats for displaying one of the following. (Default is set by the OUTPUT Command).

- H - Hexadecimal
- D - Decimal
- A - ASCII
- C - CAN-CODE

{r}
{addr}
{!xaddr}
{!?}

(AT LEAST ONE IS REQUIRED) Parameter 2 specifies the registers or memory regions to be displayed.

r displays the contents of all registers. Specific registers can be selected using Parameter 3.

addr or **!xaddr** specifies the memory address to be displayed. A range can be specified using Parameter 3.

!? displays the size of the currently inserted extended region.

[reg...]
[-addr]
[-xaddr]
[count]

(OPTIONAL) Parameter 3 used with Parameter 2 defines the specific areas to be displayed.

reg displays specific register names. PR and CC can be used as register names.

- PR specifies present location.
- CC specifies condition codes.

A maximum of eight register names can be specified.

-addr/count or **-xaddr/count** specifies the memory address range to be displayed.

EXIT

Leave The DEBUG/DEB32 Processor

The EXIT Command leaves the DEBUG/DEB32 processor.

As part of the exit processing, an end-of-file is written on the DO file. If the DO file is assigned to a spooled printer, this releases it for printing.

SYNTAX

E[XI]

Start (Or Restart) Program Execution

The GOTO Command starts or restarts program execution.

SYNTAX

G[OT] [addr]

[addr] (OPTIONAL) Parameter 1 is the instruction location to start program execution. **addr** can be "L", signifying the user program entry point address. If **addr** is omitted, execution commences from the current value of the PR register (default).

If a trace was active, it is still active when execution continues, unless it was explicitly Cleared or Reset.

EXAMPLE

```
G L
```

The special parameter "L", as shown in the example, represents the program's entry point. The absence of a parameter causes execution to continue from the current set location to the load time entry point.

HELP

Display Information About The Use Of DEBUG/DEB32

The HELP Command displays a list of the available DEBUG/DEB32 commands. The following syntax or a question mark is acceptable.

SYNTAX

H[ELP|?]

? (OPTIONAL) Parameter 1 specifies the available DEBUG/DEB32 commands. The default refers the reader to this manual.

Set Input Modes

The INPUT Command sets input modes. The INPUT format defined at DEBUG/DEB32 startup designates relative addresses without the $, and specifies all numeric values as decimal.

SYNTAX

INP {REL | ABS | DEC | HEX}

{REL | ABS | DEC | HEX} (AT LEAST ONE IS REQUIRED) Parameter 1 specifies one of the following:

- **ABS** - addresses are absolute unless $ is used.
- **REL** - addresses are relative even without $. (Default)
- **DEC** - input numbers are decimal unless preceeded by a #, in which case they are hexadecimal. (Default)
- **HEX** - input numbers are hexadecimal unless preceeded by a period (.), in which case they are decimal.

Get A Program To Be Debugged

The LOAD Command prepares a program from a file for debugging.

SYNTAX

LOA progname [,filename] [,opt]...

progname — (REQUIRED) Parameter 1 specifies the load module to be debugged.

[filename] — (OPTIONAL) Parameter 2 represents the file that contains the load module. If Parameter 2 is omitted, the default LM is used for DEBUG, and VLM is used for DEB32.

[opt]... — (OPTIONAL) Parameter 3 represents option specifiers, (Refer to the MAX IV/MAX 32 NONRESIDENT JOB CONTROL AND BATCH FACILITIES Programmer's Reference Manual that describes the $EXECUTE Directive.) The LOAD Command format is identical to that of the $EXECUTE Directive, except for the keyword.

NOTE: Unlike the Job Control usage, the LINK option cannot be specified. Program options (POPTIONS) can be specified in the P form.

All option setup is performed as if the load was a $EXECUTE. When loaded, the program is relocated by the current value of the load bias. If the program was not catalogued as relocatable, an error message is displayed and the module is not usable.

The NOC option must be used if an absolute module is to be debugged. Refer to section "Loading and Biases" in Chapter 1 of this manual.

EXAMPLE

```
LOA MYPROG,TM,NOBO,$FF
```

Loads program MYPROG from file TM. Resets the BO option and sets up the $FF option the same as it would be set up by Job Control.

The LOAD Command does not set the user program into execution, but returns to the command state after loading it.

Set (Or Display) The Load Bias

The LOCATE Command sets or displays the load bias.

SYNTAX

LOC [?]
[absaddr]

[?]
[absaddr] (OPTIONAL) Parameter 1 specifies one of the following:

? requests display of the current load bias as an absolute address.

absaddr sets the load bias to the absolute address specified by this parameter. Absaddr must be greater than the size of DEBUG/DEB32 (displayed at startup), unless the NOC option is set.

(Default) If neither parameter is used, the load bias is reset to the end of DEBUG/DEB32 (the initial condition).

MODIFY

Modify The Contents Of Register(s) Or Memory

The MODIFY Command changes the contents of registers or memory.

SYNTAX

```
        R[EG]    reg   val...
M[OD]   !xaddr   val   [val ]...
        addr     val
                 mnem[*] field 1 [field 2]...
```

R[EG]
!xaddr
addr (OPTIONAL) Parameter 1 specifies one of the following:

The **R[EG]** parameter and the pairs following it modify the named register to the provided value.

NOTE: Only numeric registers can be specified; PR and CC are invalid.

!xaddr (the extended address form) sets consecutive extended locations from **xaddr** to the values entered (CLASSIC 78xy only).

addr sets normal memory locations to the values specified. In this form of the command, a patch assembler is available.

mnem specifies the instruction mnemonic. Indirect addressing is indicated by the * parameter. Privileged mnemonics are not recognized.

Instructions HOPS, BLT, REX, NOP, and EXMA require one **field** value, that specifies the operative registers. All other instructions require two **field** values. Either field value can be null. Be aware of the fields that are significant to a given instruction. (Refer to Example 1 below.)

For DEB32 to modify an instruction containing an address, one or two parameters must be specified, depending on whether the address value is an absolute address or a relocatable address. (Refer to Example 2 below.)

EXAMPLE 1

```
LDI,4,,#FFF0
SCCC,,2
BCS,,,$#124
```

Instructions having a null first field and requiring a second field are:

BL??	SCCC
BX??	BRX
SR??	BRU

The CLM and CLMD instructions have a null first field and an optional index field. The conditional branch (B??) instructions have two null fields.

EXAMPLE 2 (DEB32 only)

If a relative address is required, only one parameter is indicated. If an absolute address is required, two 16-bit parameters are required. For example:

M 3 LAI,4,0,-1,-1

Absolute value goes in as two 16-bit quantities.

M 3 LAI,4,0,$3

An address is entered as one entry.

EXAMPLE 3

M $#101 #ABCD,@FIL,1234,$#23

places	into
#ABCD (hexadecimal)	relative #101
@FIL (CAN)	relative #102
1234 (decimal)	relative #103
$#23 (Relative address)	relative #104

NOTE: If addresses that are supplied as modifiers are to be relative, the $ flag must be specified regardless of whether the IN(PUT) REL(ATIVE) option is set or not.

NOP

No Operation

The NOP Command has two uses. The $NO form is generated through the use of Job Control procedure conditionals. Job Control procedures can be written to setup debugging sessions.

The NOP form can be used directly by the user. Either form can be used to generate comments on the DO file if the ECHO option is active.

SYNTAX

```
NOP
$NO
```

Set DEBUG/DEB32 Options

The OPTION Command specifies DEBUG/DEB32 options to be set.

SYNTAX

O[PT] [?]
[opt...]

[?]
[opt...] (OPTIONAL) Parameter 1 specifies one of the following:

? requests a display of the current setting of OPT, INP, and OUT modes.

opt represents a current option setting.

Each parameter can be (defaults are underlined):

FLO -trap floating point exceptions
NOF -don't trap floating point exceptions

PRO -issue a prompt for commands
NOP -don't issue prompts

ECH -echo command input and messages to DO
NOE -send to DO only explicitly sent output (DIS, SNA, and trace output)

CHE -check safety of input address parameters and require relocatable modules
NOC -don't check (used for loading absolute modules)

EXAMPLE

OPT ?

Displays current setting.

OUTPUT

Set Output Modes

The OUTPUT Command specifies output modes to be set.

SYNTAX

```
OUT  {ABS}
     {REL}
     {ONE}
     {TWO}
     {INS}
     {HEX}
     {DEC}
     {ASC}
     {CAN}
```

(AT LEAST ONE IS REQUIRED) Parameter 1 represents a current output mode. Defaults are underlined.

{ABS}	**ABS** - displayed addresses are absolute
{REL}	**REL** - displayed addresses are relative
{ONE}	**ONE** - trace output format is single lines for DEBUG, two lines for DEB32
{TWO}	**TWO** - trace output format is split into two lines for DEBUG, three lines for DEB32
{INS}	**INS** - trace output format doesn't include registers
{HEX}	**HEX** - TYPE/DISPLAY/SNAP default output format is hexadecimal
{DEC}	**DEC** - TYPE/DISPLAY/SNAP default output format is decimal
{ASC}	**ASC** - TYPE/DISPLAY/SNAP default output format is ASCII
{CAN}	**CAN** - TYPE/DISPLAY/SNAP default output format is CAN

Set A Breakpoint In The User Program

The PAUSE Command sets a breakpoint.

SYNTAX

P[AU] addr [count]
[param a param b]

addr (REQUIRED) Parameter 1 specifies the address for a breakpoint to occur.

[count]
[param a param b] (OPTIONAL) Parameter 2 specifies a condition for a breakpoint or an evaluation to occur. (Default = An unconditional breakpoint is set at Parameter 1 if neither of the following alternatives is chosen.)

count specifies the breakpoint that is taken every **count**-th time that Parameter 1 is encountered.

param a param b pair is evaluated each time Parameter 1 is encountered. Values are represented by a constant value (hexadecimal, decimal, or CAN), a register identifier, a bit mask, or a memory address (extended memory (DEBUG only) or normal memory).

In each case, except the bit mask, the two parameters are checked for equality. If they are equal, the breakpoint is taken.

In the case of the bit mask, the bit mask and the other parameter are bit-wise ANDed, and the result is compared to the mask value for equality. If it is equal, the breakpoint is taken.

EXAMPLES

P $#234
- Regains control at address #234 in program.

PAU $#123 5
- Specifies that only every fifth time that the instruction at location #123 is encountered should a breakpoint take place.

P $#456 R3 M $#7A2
- Each time the instruction at location #456 is encountered, the contents of register 3 and the contents of memory location #7A2 are compared. If they are equal, a breakpoint takes place.

Set A Reference Trace

The REFERENCE Command sets a reference trace.

In addition to the trace line that is output, the following message is output to the DO file on a match; REF FOUND.

SYNTAX

REF addr [-addr][addr 2]
[count][!xaddr]

addr (REQUIRED) Parameter 1 specifies the start address for the trace.

[-addr]
[count] (OPTIONAL) Parameter 2 specifies the end address for the trace.

[addr 2]
[!xaddr] (OPTIONAL) Parameter 3 specifies that output is to be generated only for instructions that reference (as an operand) Parameter 3.

EXAMPLE

REF $#87-#114 #ABC

-Specifies a trace to be set from location #87 to #114. Trace output lines are generated only for those instructions that reference absolute address #ABC. In addition to the trace output, the message REF FOUND is output to the DO file. The reference address of a REFERENCE TRACE is not directly involved in execution control. It can be specified in several REFERENCE TRACES, and can be the same address as specified for a control address.

Deactivate All Active Traces

The RESET Command unconditionally deactivates all active traces without removing them from the DEBUG/DEB32 table.

SYNTAX

R[ES]

Set A Memory Array To A Constant

The SET Command sets a specified memory array to a constant value. Extended sets are analogous, with the extended flag specified, and 21-bit addresses are permitted.

SYNTAX

```
SET  [addr]     [-addr]     value
     [!xaddr]   [count]
                [-xaddr]
                [count]
```

[addr]
[!xaddr] - (OPTIONAL) Parameter 1 specifies the start address of the array of memory or extended memory to be set to a constant value.

[-addr]
[count]
[-xaddr]
[count] - (OPTIONAL) Parameter 2 specifies either the end address or the location count for the array of memory or extended memory to be set to a constant value.

value - (REQUIRED) Parameter 3 specifies a numeric or CAN-CODE value for the array specified in Parameter 1 through Parameter 2. Unallocated memory portions are set to this value up to the first unallocated location. A message is then displayed and processing is stopped.

Extended arrays are completely checked before the setting starts and therefore are never partially set.

EXAMPLES

```
SET $#100-#17F 0
```
-Sets relative locations #100 through #17F to zero.

```
SET $#234 47 #FFFF
```
-Sets a 47 word array starting at relative location #234 to #FFFF.

```
SET !#2FFFF 2000 #2020
```
-Sets a 2000 word array in the extended region, starting at location #2FFFF, to #2020.

Display DEBUG Or DEB32 Table Contents

The SHOW Command displays the contents of the DEBUG or DEB32 table.

SYNTAX

SHO [P]
[S]
[T]
[B]
[R]

[P]
[S]
[T]
[B]
[R] (OPTIONAL) Parameter 1 specifies one of the following:

P represents Pause.
S represents Snaps.
T represents Traces.
B represents Btraces.
R represents Reference traces.

(Default) If no parameter is specified, display all DEBUG/DEB32 table entries.

Set A Snapshot

The SNAP Command sets a snapshot. All SNAP output is directed to the DO file.

SYNTAX

```
S[NA]  [format]  addr1  [addr 2      [-addr 3]]
                         [count  ]

                         [!xaddr     [-xaddr 3]]
                         [count  ]
```

format — (OPTIONAL) Parameter 1 displays all values in the specified format. The default is set at the time the SNAP Command is entered if format is not specified. Formats provided are:

H-hexadecimal
D-decimal
A-ASCII
C-CAN (actually, unCAN)

addr 1 — (REQUIRED) At Parameter 2 a snapshot is set. If the address ranges in Parameter 3 through Parameter 4 are omitted, at snap time display only the registers.

[addr2]
[-addr3]
[!xaddr2]
[-xaddr3]
[count] — (OPTIONAL) Parameters 3 and 4 represent an address range. If specified, they display the registers and the contents of the memory range.

EXAMPLES

S $#123
- Displays the registers when the user program executes location #123. Execution continues, since snapshots do not cause command state to be entered.

S $#234 $#4C3 44
- Displays the registers and the contents of memory from location #4C3 for 44 words, when the user program executes location #234.

Step Through The Program

The STEP Command steps through the program and displays a single text line from the current or specified location.

SYNTAX

```
STE   ?
[STE] [count [addr] ]
```

?
[count] (OPTIONAL) Parameter 1 specifies the number of instructions to step.

? requests the display of a text line showing the headings for the STEP output.

count single steps a specified number of instructions starting at Parameter 2.

[addr] (OPTIONAL) Parameter 2 specifies the starting address. **addr** can be "L", signifying the user program entry point address.

If Parameter 2 is omitted, the stepping commences from the current location.

NOTE: If a null input line or just a carriage return is input, a single step from the current location is taken.

EXAMPLES

3 $#123

-Steps three instructions starting at location #123.

2

-Steps two instructions starting at the current location.

CR (carriage return)

-Steps a single instruction from the current location.

Set A Trace

The TRACE Command sets a normal trace, starting at specified address and running until specified end address or for specified number of words.

SYNTAX

TRA start-addr {-end-addr / count}

start-addr (REQUIRED) Parameter 1 represents the start address for a normal trace.

{-end-addr / count} (AT LEAST ONE IS REQUIRED) Parameter 2 represents the end of a trace.

-end-addr signals the end of the run.

count specifies the number of words the trace contains.

EXAMPLE

TRA $#123-#1C2

-Specifies that trace is to start when (and if) the program executes the instruction at #123. Trace display lines are to be generated for each subsequent instruction until the instruction at location #1C2 is executed. At that time, normal (non-tracing) execution resumes if no other traces are active.

Display Values On The CO File

The TYPE Command displays values on the CO file. The syntax and semantics of the TYP Command are identical to the DIS Command, except for the keyword, and the fact that output goes to the CO file, rather than the DO file. Displaying extended memory is identical to displaying normal memory except that the extended flag (!) is given, and addresses can be up to 21 bits, instead of 16 bits.

SYNTAX

```
T[YP]  [format]  {r}        [reg...]
                 {addr}     [-addr]
                            [count]
                 {!xaddr}   [-xaddr]
                            [count]
                 {!?}
```

[format] (OPTIONAL) Parameter 1 specifies one of the following formats for displaying:

H - Hexadecimal
D - Decimal
A - ASCII
C - CAN-CODE

The default is set by the OUTPUT Command.

{r}
{addr}
{!xaddr}
{!?} (AT LEAST ONE IS REQUIRED) Parameter 2 specifies the registers or memory regions to be displayed.

r displays the contents of all registers. Specific registers can be selected using Parameter 3.

addr or **!xaddr** specifies the memory address to be displayed. A range can be specified using Parameter 3.

!? displays the size of the currently inserted extended region.

[reg...]
[-addr]
[count]
[-xaddr]
[count]

(OPTIONAL) Parameter 3 used with Parameter 2 specifies the specific areas to be displayed.

reg displays specific register names. PR and CC can be used as register names.

- PR specifies present location.
- CC specifies condition codes.

A maximum of eight register names can be specified.

-addr/count or **-xaddr/count** specifies the memory address range to be displayed.

EXAMPLES

(Displaying **memory** with the TYPE Command)

T $#123 - Displays the contents of relative location #123 on the terminal.

T $#123-#147 - Displays locations #123 through #147.

T $#123 20 - Displays 20 locations starting at #123.

(Displaying **registers** with the TYPE Command)

T R - Displays all registers (R0-R15, the program register, and the condition codes). Two registers can follow the R as shown in the next example.

T R 1 3 CC 7 - Displays registers 1, 3, the condition codes, and register 7, in that order. A maximum of eight registers can be specified in a single command.

(Displaying **extended memory** with the TYPE Command (**DEBUG only**))

T !123-456 - Displays the contents of addresses 123 through 456 (decimal) in the extended region.

CHAPTER 4
MESSAGES

DEBUG/DEB32 generate a variety of messages for errors and expected conditions. Errors can occur on command processing and during execution state. An attempt is made to display an error pointer under the command line in command errors to localize the error.

The following is an alphabetical list of the messages with a brief explanation. As much as possible, the messages are self-explanatory.

ADDRESS ALREADY IN DEBUG TABLE

Meaning: The user has attempted to make an entry in the debug table that matches the address of an entry that was already there.

Recommended Action: Enter command with a different address.

ADDRESS IS NOT A TRACE TYPE ENTRY

Meaning: The user has attempted to activate an address that makes a pause or snap. Only trace type addresses can be activated.

Recommended Action: Enter a trace type address.

ADDRESS NOT ALLOCATED

Meaning: The user has attempted to modify a location that is not in allocated memory. This message is given only if the first address of the modify is unallocated. If the list crosses into unallocated memory, no message is given; however, the rest of the list is effectively ignored.

Recommended Action: Modify locations in allocated memory.

ADDRESS NOT IN DEBUG TABLE

Meaning: The user has attempted to activate or clear an address that is not in the debug table.

Recommended Action: Activate or clear an address in the debug table.

ADDRESS OUT OF BOUNDS

Meaning: The user has specified an address in a command that is out of the valid address bounds, is unallocated, or is within debug.

Recommended Action: Specify a valid allocated address.

ATTEMPT TO MODIFY MEMORY PAST ADDR #FFFF

Meaning: The user has attempted to modify memory by wrapping around the end.

Recommended Action: Modify memory within valid boundries.

CANNOT HAVE TWO "B" TYPE OPERANDS

Meaning: In an equal type pause, the user attempted to specify two bit mask type operands, which is illegal.

Recommended Action: Specify only one "B" type operand.

CURRENT LOAD BIAS: #addr

Meaning: The current bias used for loading overlays with the load command is addr. Current relocation bias: #addr.

Recommended Action: No response needed.

*DEBUG REV E.1 LOAD BIAS addr

Meaning: This is the startup message, which shows the initial load bias.

Recommended Action: No response needed.

DEBUG TABLE FULL

Meaning: The user has attempted to make more than 20 simultaneous entries in the debug table.

Recommended Action: Some entries must be removed with the clear command before attempting to add more.

EFFECTIVE ADDRESS OUT OF BOUNDS AT #addr

Meaning: While stepping or tracing, the effective address is out of the valid addressing bounds. The current PR is addr.

EXIT ATTEMPTED AT #addr

Meaning: A REX exit was executed at addr.

Recommended Action: Check addr.

EXTENDED ADDRESS OUT OF BOUNDS AT #addr

Meaning: While stepping or tracing the extended address is out of bounds. The current PR is addr.

Recommended Action: Check addr.

FLOATING POINT EXCEPTION AT #addr

Meaning: If stepping or tracing, a floating point overflow or underflow has occurred at addr (or if not stepping or tracing, at the instruction prior to the one at addr).

Recommended Action: Check addr.

ILLEGAL INSTRUCTION AT PAUSE OR SNAP

Meaning: An illegal instruction exists at the location of a pause or snap. The illegal instruction was detected at the time the snap was encountered, or the time of the GOTO following the pause.

Recommended Action: Check instruction at pause or snap.

ILLEGAL OR UNTRACEABLE REX AT #addr

Meaning: While tracing or stepping, a REX was encountered that cannot be executed.

Recommended Action: Check the illegal REX located at **addr.**

INVALID ADDRESS PARAMETER

Meaning: The user has supplied a command parameter that must be an address but cannot be legally interpreted as one.

Recommended Action: Enter valid address parameter.

INVALID COMMAND

Meaning: The user has supplied a command name that is not one of the valid debug commands.

Recommended Action: Supply valid command name.

INVALID FILE/DEVICE NAME OR I/O ERROR

Meaning: The user supplied an invalid file or device name for a file command or an I/O error occurred during the execution of the file command.

Recommended Action: Supply a valid file or device name.

INVALID FILE NAME

Meaning: The user supplied a file name on a load command that is not valid.

Recommended Action: Supply valid file name.

INVALID FORMAT SPECIFIER

Meaning: The user has specified a display format that is not valid.

Recommended Action: Specify valid display format.

INVALID INSTRUCTION MNEMONIC

Meaning: The user has supplied a mnemonic in a modify command that is not valid.

Recommended Action: Supply a valid mnemonic.

INVALID KEYWORD PARAMETER

Meaning: The user has specified a keyword type parameter that is not valid at that point.

Recommended Action: Supply a valid keyword type.

INVALID MODULE NAME

Meaning: The user has supplied a load module name in a load command that is not valid.

Recommended Action: Supply a valid load module name.

INVALID NUMERIC PARAMETER

Meaning: The user has supplied a numeric or address parameter that is not valid.

Recommended Action: Supply valid numeric parameter.

INVALID OPTION SPECIFIED

Meaning: An option specified in an option, input, or output command is not valid.

Recommended Action: Enter valid options in option, input and output commands.

INVALID REGISTER PARAMETER

Meaning: A specified register parameter is not a number in the range 0 to 15.

Recommended Action: Enter valid address register parameters from 0 to 15.

LOAD ERRORS. MODULE NOT USABLE

Meaning: The module loaded by the load command encountered loader errors. Error messages from the loader should have been displayed also.

Recommended Action: Exit or try a different load module. (Note: The debug table has been cleared.)

*MAX32 DEBUG A.0 LOAD BIAS addr

Meaning: This is the startup message, which shows the initial load bias.

Recommended Action: No response needed.

MISSING PARAMETER

Meaning: An expected parameter was not found in the command line.

Recommended Action: Enter parameter in command line.

MODULE NOT FOUND

Meaning: The specified module was not located on the specified load module file.

Recommended Action: Check specified module and load module file for validity.

MODULE NOT RELOCATABLE

Meaning: The module was not relocatable and the CHE option was set.

Recommended Action: The module must be fixed or made relocatable. (Note: The debug table has been cleared.)

NO EXTENDED MEMORY REGION INSERTED

Meaning: An extended memory operation was requested but no extended memory region has been inserted for the current task.

Recommended Action: Insert extended memory region for current task.

NO MODULE LOADED

Meaning: A GOTO or step command was attempted, but no module has successfully been loaded.

Recommended Actions: Verify module name in command. Verify existence of module.

OVERLAPPING ADDRESS RANGE

Meaning: A range of addresses was specified in which the low bound exceeds the high bound.

Recommended Action: Specify valid address range.

PAUSE AT #addr

Meaning: Breakpoint has been taken at **addr.**

Recommended Action: No response needed.

POSSIBLE INTERNAL ERROR AT #addr

Meaning: An error has occurred that may indicate an error in the debug processor.

Recommended Action: Please save all information about the debugging session and contact MODCOMP.

PROCESSING STOPPED. MEMORY UNALLOCATED AT #addr

Meaning: During the processing of a set command, an unallocated block of memory was encountered. Processing of the command stops at **addr** but all memory before that point has been set.

Recommended Action: Allocate memory. Process the set command.

PROTECT VIOLATION AT #addr

Meaning: A protect violation (level 2 trap) has occurred at **addr.**

Recommended Action: Check address.

REX DEBUG NOT IN TABLE

Meaning: Debug did not expect this special REX, whose use is reserved for the Debug processor. If this instruction (REX, #1F) was not accidentally or intentionally placed in the program by the user, this message may indicate an internal debug error.

Recommended Action: If this occurs, please save all information about the debugging session and contact MODCOMP.

REX "USE" OR "USERTRAP" AT #addr

Meaning: A REX use or usertrap was encountered in the user's program at **addr.**

Recommended Action: Debug cannot process these REX calls.

UNCANNABLE PARAMETER

Meaning: A parameter that was preceeded by the can flag (@) was not CAN codable.

Recommended Action: Specify a CAN codable parameter.

UNIMPLEMENTED INSTRUCTION AT #addr

Meaning: An unimplemented instruction (UIT) was encountered in the user's program at **addr.**

Recommended Action: Check instruction at specified address.

UNIMPLEMENTED REX AT #addr

Meaning: An unimplemented instruction (UIT) was encountered in the user's program at **addr.**

Recommended Action: Check instruction at specified address.

UNTRACEABLE INSTRUCTION AT #addr

Meaning: An instruction that debug cannot trace (Example: A privileged instruction) was encountered while tracing or stepping. The instruction is located at **addr.**

Recommended Action: Check the instruction at the specified address.

CHAPTER 5
OUTPUT FORMATS

Chapter 5 summarizes the output of the various DEBUG/DEB32 functions. The output is as self-explanatory as possible. A set of output examples with a commentary is provided.

A Type or Disp all registers (hexadecimal format):

```
(REG 0-7)   :  0961   E7A0   2900   ED20   0000   230B   01D0   0000
(REG 8-15)  :  0000   0000   0000   0000   0000   0000   0000   0000
(PR & CC)   :  0000   0000
```

A similar display but in ASCII format. Note that values that do not have printable representations are left in hexadecimal (byte by byte).

```
(REG 0-7)   :  0004   0000    A B    C !   0000    xD0   01D0   00 J
(REG 8-15)  :   <>     @00   0000   0000    S C     A      00   0000
(PR & CC)   :  0002   0000
```

The following display is produced with the following command:

```
D D R 2 4 7
```

The display is of those three registers only, in that order, and in decimal format.

```
(REGS)  :     4660          0          0
```

A memory display, starting at 0 and going up to #13, in hexadecimal format:

```
(0000)   :  ED20   1234   6D56   0E13   F77C   0000   0001   0002
(0008)   :  0003   0004   0005   0006   0007   0008   0009   ED80
(0010)   :  0012   2332   E545   2900
```

Memory from #14 through #23, in CAN format:

```
(0014)   :  JQZ    J3T    KCN    KPH    K2B    LA9    LN3    LOX
(001C)   :  M R    MML    MZF    M$     NK7    G77    HG1    HTV
```

EXAMPLE (of DEB32 memory display)

```
MAX32 DEBUG        REV.  A.0        DATE 22 MAR 85   08:37:54   PAGE  19
LAST MODULE LOADED BY DEBUG  :  DBT1

( 00000000):  ED40  8014  6077  E537  0000  3A0E  E637  0000
( 00000000):  3A22  607F  714F  0000  3A03  2312  0001  0002
( 00000000):  0003  0004  0005  0006  0007  0008  0009  000A
( 00000000):  0640  0080  1200  4420  4520  000A  000B  FFFF
( 00000000):  0013  0014  0001  0002  0003  0004  0005  0006
( 00000000):  0007  0008  0009  000A  0640  0C80  12C0  4420
( 00000000):  4520  000A  000B  FFFF  0C13  0014  0000
```

An extended memory display is similar, but the address field (between the parenthesis) is six hexadecimal digits, rather than four.

The STEP header line (that can be displayed with the STE ? command):

```
ADDR  INST   MNEM, R,X    EADD  (EAB)  (EAA)  NZOC  IOU
```

ADDR is the address of the instruction; INST is the hexadecimal value of the instruction; MNEM is its mnemonic, if it is indirect a * follows it. R and X are the two register fields; EADD is the effective address; (EAB) and (EAA) are the contents of EADD before and after execution of the instruction; NZOC are the condition codes; and IOU are Integer overflow history, floating point Overflow, and floating point Underflow, respectively. If an error occurred, a star (*) is displayed in column 1.

Output of a step 5 command (this format is also used for an INS formatted trace):

```
0000   ED20  LDI   2              C234   C234   N...   ...
0002   6D56  TRR   5,6                          .Z..   ...
0003   0E13  LCCC  1                            .Z..   ...
0004   F77C  HOP   $-4      0000                .Z..   ...
0000   ED20  LDI   2              C234   C234   N...   ...
```

EXAMPLE (of trace output (ONE format, but for the purposes of this manual truncated to 80 columns), with a trace output page heading)

```
MAX IV DEBUG        REV.  E.0     DATE  7/12/84      17:04:28      PAGE  3
LAST MODULE LOADED BY DEBUG : TEST

ADDR INST  MNEM,R,X EADD (EAB)  (EAA)  NZOC IOU  R1    R2    R3    R4    R5
0000  ED20 LDI   2          C234  C234  M...  ...  E7A0  C234  ED20  0000  2308
0002  6D56 TRR   5,6                     .Z..  ...  E7A0  C234  ED20  0000  0000
0003  0E13 LCCC  1                       .Z..  ...  0004  C234  ED20  0000  0000
0000  ED20 LDI   2          C234  C234  N...  ...  0004  C234  ED20  0000  0000
```

EXAMPLE (output of the SHO P command)

```
PAUSES
ADDRESS TYPE   INIT OR PAR 1    CUR OR PAR 2
0002  SMP
#0003   CNT        17            17
#0009   EQU       064C       R     4
#0014   EQU     B  FF0F        M !0001CF
```

TYPE is SMP for simple pauses, CNT for counted pauses, and EQU for equal type pauses. The first number for CNT types is the count specified in the PAUSE Command, the second is the current count (times remaining until the breakpoint is taken). For EQU type pauses, PAR 1 and PAR 2 are the same as were specified in the PAUSE Command.

EXAMPLE (output of the SHO T command)

```
TRACES
 START    END     ACTIVE
 #0000    #0010    N
```

START and END are self-explanatory, and ACTIVE indicates whether or not the trace is currently active. A SHO B is identical.

EXAMPLE (output of the SHO R command)

```
REFERENCE TRACES
START    END     ACTIVE  REF ADDRESS
#0000    #0010    Y       007B
```

REF ADDRESS is the reference address specified in the Reference Trace Command.

EXAMPLE (output of the SHO S command)

```
SNAPS

SNAP ADDR   X   LOW     HIGH
#0008
#000D           007B    0091
```

LOW and HIGH represent the low and high addresses of the memory block to be displayed. If these addresses are extended, a ! is shown under the X column. If no memory is to be displayed, these are blank.

EXAMPLE (output of the OPT ? command)

```
      CURRENT   OPTIONS   :CHE   FLO    PRO   ECH
                INPUT     :REL   DEC
                OUTPUT    :REL   ONE    HEX
```

EXAMPLE (output of DEB32 trace (TWO format), with a trace output page heading)

```
MAX32 DEBUG    REV.  A.0     DATE 22 MAR 85  08:34:50     PAGE 8
LAST MODULE LOADED BY DEBUG  :  DBT1

ADDR    INST    MNEM  R,X EADD      (EAB)        (EAA)        NZOC IOU
R1        R2          R3          R4          R5          R6          R7
R8        R9          R10         R11         R12         R13         R14         R15

00000000  ED40  LDI   4              0014        0014        .... ...
00000000  00003A00  00000000  00000014  00000000  00000100  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

00000002  6C77  XOR   7,7                                   .Z.. ...
00000000  00003A00  00000000  00000014  00000000  00000100  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

00000003  E537  LDM   3,7 0000000E  0001        0001        .... ...
00000000  00003A00  00000001  00000014  00000000  00000100  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

00000006  E637  STM   3,7 00000022  0000        0001        .... ...
00000000  00003A00  00000001  00000014  00000000  00000100  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

00000009  607F  ABR   7,F                                   .... ...
00000000  00003A00  00000001  00000014  00000000  00000100  00000001
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

0000000A  714F  SBRB  4,F 00000003                          ...C ...
00000000  00003A00  00000001  00000013  00000000  00000100  00000001
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

00000003  E537  LDM   3,7 0000000F  0002        0002        ...C ...
00000000  00003A00  00000002  00000013  00000000  00000100  00000001
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

00000006  E637  STM   3,7 00000023  0000        0002        ...C ...
00000000  00003A00  00000002  00000013  00000000  00000100  00000001
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

0000000A  607F  ABR   7,F                                   .... ...
00000000  00003A00  00000001  00000013  00000000  00000100  00000002
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

00000003  714F  SBRB  4,F 00000003                          ...C ...
00000000  00003A00  00000002  00000013  00000000  00000100  00000002
```

EXAMPLE (output of a DEB32 trace (INS format) with a trace output page heading)

```
MAX32 DEBUG    REV.  A.0     DATE 22 MAR 85   08:36:50     PAGE 17
LAST MODULE LOADED BY DEBUG  :  DBT1

ADDR      INST  MNEM  R,X  EADD        (EAB)     (EAA)     NZOC IOU
00000000  ED4C  LDI   4                 0014      0014      .... ...
00000002  6C77  XOR   7,7                                   .Z.. ...
00000003  E537  LDM   3,7  0000000E     0001      0001      .... ...
00000006  E637  STM   3,7  00000022     0000      0001      .... ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  0000000F     0002      0002      .... ...
00000006  E637  STM   3,7  00000023     0000      0002      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000010     0003      0003      ...C ...
00000006  E637  STM   3,7  00000024     0000      0003      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000011     0004      0004      ...C ...
00000006  E637  STM   3,7  00000025     0000      0004      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000012     0005      0005      ...C ...
00000006  E637  STM   3,7  00000026     0000      0005      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000013     0006      0006      ...C ...
00000006  E637  STM   3,7  00000027     0000      0006      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000014     0007      0007      ...C ...
00000006  E637  STM   3,7  00000028     0000      0007      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000015     0008      0008      ...C ...
00000006  E637  STM   3,7  00000029     0000      0008      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000016     0009      0009      ...C ...
00000006  E637  STM   3,7  0000002A     0000      0009      ...C ...
00000009  607F  ABR   7,F                                   .... ...
0000000A  714F  SBRB  4,F  00000003                         ...C ...
00000003  E537  LDM   3,7  00000017     000A      000A      ...C ...
00000006  E637  STM   3,7  0000002B     0000      000A      ...C ...
00000009  607F  ABR   7,F                                   .... ...
```

EXAMPLE (output of DEB32 trace (ONE format, but for the purposes of this manual truncated to 80 columns), with a trace output page heading)

```
MAX32 DEBUG   REV.  A.0    DATE 22 MAR 85  08:30:23    PAGE   11
LAST MODULE LOADED BY DEBUG  :  DBT1

ADDR      INST  MNEM  R,X EADD      (EAB)     (EAA)     NZOC IOU  R1
                                    R7        R8        R9        R10

00000000  ED40  LDI   4             0014      0014      .... ...  00000000
                                    00000000  00000000  00000000  00000000

00000002  6C77  XOR   7,7                                .Z.. ...  00000000
                                    00000000  00000000  00000000  00000000

00000003  E537  LDM   3,7  0000000E  0001      0001      .... ...  00000000
                                    00000000  00000000  00000000  00000000

00000006  E637  STM   3,7  00000022  0001      0001      .... ...  00000000
                                    00000000  00000000  00000000  00000000

00000009  607F  ABR   7,F                                .... ...  00000000
                                    00000001  00000000  00000000  00000000

0000000A  714F  SBRB  4,F  00000003                      ...C ...  00000000
                                    00000001  00000000  00000000  00000000

00000003  E537  LDM   3,7  0000000F  0002      0002      ...C ...  00000000
                                    00000001  00000000  00000000  00000000

00000003  E637  STM   3,7  00000023  0000      0002      ...C ...  00000000
                                    00000001  00000000  00000000  00000000

00000009  607F  ABR   7,F                                .... ...  00000000
                                    00000002  00000000  00000000  00000000

0000000A  714F  SBRB  4,F  00000003                      ...C ...  00000000
                                    00000002  00000000  00000000  00000000

00000003  E537  LDM   3,7  00000010  0003      0003      ...C ...  00000000
                                    00000002  00000000  00000000  00000000

00000006  E637  STM   3,7  00000024  0000      0003      ...C ...  00000000
                                    00000002  00000000  00000000  00000000

00000009  607F  ABR   7,F                                .Z.. ...  00000000
                                    00000003  00000000  00000000  00000000

0000000A  714F  SBRB  4,F  00000003                      .C.. ...  00000000
```

INDEX

!, 1-27
$ flag, 1-5

ABORT, 1-10
absolute address, 1-3
ABSOLUTE, 1-3
ACTIVATE Command, 1-9
ACTIVATE, 3-2
AI, 2-1
ALLOCATE, 1-10
ASCII, 1-5
ASSIGN Command, 2-1
ASSIGN, 2-1
AVFILE, 2-1
AVRECORD, 2-1

BIAS Command, 1-3
BIAS, 3-3
BKFILE, 2-1
BKRECORD, 2-1
BLT, 3-14
BREAKPOINTING, 1-6
BTRACE Command, 1-9
BTRACE, 1-6, 3-4
bugs, 1-1

CAN (unCAN), 1-5
CI, 2-1
CLASSIC II/75, 1-5
CLEAR, 3-5
CO, 2-1
Command state, 1-2
CONCEPT ERRORS, 1-1

DEALLOCATE, 1-10
DEB32 (MAX 32), 1-1
DEBUG (MAX IV), 1-1
DEBUG/DEB32, 1-1
debugging, 1-1
decimal, 1-5
DEFINE, 3-6
DELAY, 1-5
DESIGN ERRORS, 1-1
DISPLAY Command (DIS or D), 1-5
DISPLAY, 3-7
DO, 2-1

Error Processing Addresses (EPAs), 1-10
Execution state, 1-2
EXIT, 3-8
EXMA, 3-14
extended flag (!), 1-5
extended memory flag, 1-14

file manipulation commands, 2-1
files (used by DEBUG/DEB32), 2-1

GOTO Command, 1-7
GOTO, 3-9

HELP, 3-10
hexadecimal, 1-5
HOME, 2-1
HOPS, 3-14

IMPLEMENTATION ERRORS, 1-1
INPUT, 3-11
INSGLO, 1-10

Job Control overlays, 1-1

keywords, 1-2

LOAD Command (LOA), 1-4
LOAD Command, 1-7, 2-1
LOAD, 3-12
LOCATE, 3-13
LOVER, 1-10

machine language programs, 1-1
MARS, 1-10
MAX 32 Debug (DEB32), 1-1
MAX IV and MAX 32 Task/Overlay Cataloger (TOC/TOC32), 1-4
MAX IV Debug (DEBUG), 1-1
MODCOMP CLASSIC, 1-1
MODIFY Command (MOD or M), 1-5
MODIFY Command, 3-14
MODIFY, 3-14
NOP, 3-14

OPTION Command, 1-8, 3-17
OPTION, 3-17
OUTPUT Command, 1-9
OUTPUT Command, 3-18
OUTPUT FORMATS, 5-1

patch assembler, 1-5
PAUSE Command, 1-7
PAUSE Command, 3-19
PAUSE, 1-2

quick load, 2-1

REFERENCE Command, 3-20
REFERENCE, 1-6
relative addresses, 1-3
RELATIVE, 1-3
RESET Command, 1-9
RESET, 3-21
REWIND, 2-1
REX, 3-14

SET Command, 1-6
SET, 3-22
SHOW, 3-23
SNAP Command, 1-8
SNAP, 1-2, 3-24
STEP Command, 3-25
STEP, 3-25

TRACE Command, 3-26
TRACE, 1-6
trap, 1-7
TYPE Command, 3-27
TYPE, 1-5

WEOF, 2-1

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 3824 FT. LAUDERDALE, FL 33309

POSTAGE WILL BE PAID BY ADDRESSEE

MODULAR COMPUTER SYSTEMS
1650 W. McNAB ROAD
P.O. BOX 6099
FT. LAUDERDALE, FLORIDA 33310

Attention: TECHNICAL PUBLICATIONS, M.S. #85

Fold

MODCOMP

Please comment on the publication's completeness, accuracy, and readability. We also appreciate any general suggestions you may have to improve this publication.

If you found errors in this publication, please specify the page number or include a copy of the page with your remarks.

Your comments will be promptly investigated and appropriate action will be taken. If you require a written answer, please check the box and include your address below.

☐

Comments: _____

Manual Title _____

Manual Order Number _____ Issue Date _____

Name _____ Position _____

Company _____

Address _____

_____ Telephone ( ) _____

MCS 148D (1/83)

MODCOMP

Corporate Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310, Tel: (305) 974-1380, TWX: 310-372-7837

International Headquarters:
MODULAR COMPUTER SERVICES, Inc., The Business Centre, Molly Millars Lane, Wokingham, Berkshire, RG11 2JQ, UK, Tel: 0734-786808, TLX: 851849149

Latin American Sales Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310, Tel: (305) 975-6562, TLX: 3727852

Canadian Headquarters:
MODCOMP Canada, Ltd., 400 Matheson Blvd. East, Unit 24, Mississauga, Ontario, Canada L4Z 1N8, Tel: (416) 890-0666, TELEX: 06-961279

SALES & SERVICE LOCATIONS THROUGHOUT THE WORLD

"The technical contents of this document while accurate as of the date of publication, are subject to change without notice."

Printed in the U.S.A.